L8: Entry 1 of 1 File: USPT Oct 26, 1993

DOCUMENT-IDENTIFIER: US 5257379 A
TITLE: Establishing synchronization of hardware and software I/O configuration definitions

Application Filing Date (1):
19910904

Detailed Description Text (238):
Operating system configuration identifier 1101F (an arbitrary 8-byte string unique to a particular configuration within an IODF)

Detailed Description Text (244):
The operating system configuration identifier is the one specified by the installation for IPL or the default identifier, if none was specified and only one operating system definition exists in the IODF used for IPL.

Detailed Description Text (250):
In this case, the new IODF WWUV and the new operating system configuration identifier are stored into the configuration token. Additionally, the configuration token sequence count is incremented. If the EDT is rebuilt, then the EDT identifier is updated.

Detailed Description Text (394):
FIG. 21 is an illustration of multiple IODFs. In the illustrated embodiment, there may be a maximum of 256 IODFs. Each IODF has a suffix which is a 2-digit hexidecimal number from OO to FF which is a part of the IODF name. Although, in FIG. 21, consecutive numbers are shown, it will be understood that there is no requirement that consecutive numbers be used in the IODF suffix. Each of the IODFs contains one of the structures shown in FIG. 17A, and may contain multiple structures shown in FIG. 17B. Each of the IODFs may contain multiple PRRs, one for each processor I/O configuration definition in the IODF. As previously mentioned, each PRR includes a hardware token for identifying the processor I/O configuration definition represented by the PRR. Each FIG. 17B structure includes an OSR record which includes an operating system configuration identifier for identifying the operating system I/O configuration definition of its FIG. 17B structure.

Detailed Description Text (397):
If the result at 2206 is yes, or if 2207 is used, a check is made at 2208 to determine if both the operating system configuration identifier and the IODF suffix are specified. If no, default processing is used at 2209 (see FIG. 23), and if yes, non-default processing is used at 2210 (see FIG. 24).

Detailed Description Text (399):
FIG. 23 is a flow chart of default processing at IPL. Blocks 2301, 2302, 2303 and 2304 are a summary of the flow chart of FIG. 22. A check is made at 2305 to see if an operating system (OS) configuration identifier is specified. If no, a check is made at 2306 to determine if the IODF suffix is specified. If the result at 2306 is no, MVS searches at 2307 for an IODF that contains a single OS configuration identifier. In other words, the FIG. 17B structure (see FIG. 21) contains only one OSR record which indicates that the single operating system I/O configuration

definition in that IODF is to be used. If the result at 2307 is no, MVS enters the wait state at 2308. If the result at 2307 is yes, the IODF found at 2307 is used as the default IODF, as shown at 2312.

Detailed Description Text (400):
If the result at 2306 is yes, a check is made at 2309 to determine if the IODF specified at 2306 contains a single OS configuration identifier. If the result at 2309 is no, MVS enters the wait state at 2310.

Detailed Description Text (401):
Returning to 2305, if an OS configuration identifier was specified at 2305, MVS searches for an IODF that contains the specified OS configuration identifier at 2311. If the result at 2307, 2309, or 2311 is yes, a check is made at 2313 to determine if the hardware I/O configuration definition matches the representation of the hardware I/O configuration used by the channel subsystem. This is done by fetching the hardware token from the HSA, and searching the PRRs of the IODF for that hardware token. If the hardware token is found at 2313, both the software and hardware configuration definitions can be changed at 2314, as previously discussed. If the Hardware token is not found in the IODF, only the software configuration definition can be changed at 2315. If only the software configuration definition can be changed, a warning is issued that the hardware configuration definition is not allowed to be changed.

Detailed Description Text (402):
FIG. 24 is a flowchart showing the non-default process at IPL. Blocks 2401, 2402 and 2403 form a summary of FIG. 22. At 2405, MVS looks for the specified OS configuration identifier in that specified IODF. If the specified OS configuration identifier is not found in the specified IODF, MVS enters the wait state at 2406. If MVS does find the specified OS configuration identifier in the specified IODF, that IODF is checked at 2407 to confirm that it contains an I/O configuration definition that matches the representation used by the channel subsystem. This is done by fetching the hardware token from the HSA, and searching for it in the PRRs of the specified IODF. If a match is found in 2407, both software and hardware configuration definitions can be changed at 2408, as previously explained. If a match is not found at 2407, only the software configuration definition can be changed at 2409, and a warning is issued that changes to the hardware configuration definition are not allowed.

CLAIMS:

13. In a data processing I/O system having a main storage for storing data and data processing instructions arranged in programs including an operating system, a storage device for storing multiple I/O definition files (IODFs), each IODF having hardware configuration information for multiple processor I/O configuration definitions, each processor I/O configuration definition having a hardware token for identification, a processor controller for containing hardware configuration information for one processor I/O configuration definition and a copy of its hardware token received from a base IODF in said main storage, and a hardware storage area (HSA) connected to said processor controller for storing a hardware configuration definition established from the hardware configuration definition in said processor controller, said hardware configuration definition containing said copy of said hardware token in said processor controller, and said main storage having a software configuration definition established from said base IODF, a computer implemented method for determining if the software configuration definition corresponds to the hardware configuration definition comprising:

at a specified activate time, fetching said copy of said hardware token from said HSA; and

comparing the hardware token of said one processor I/O configuration definition of

said base IODF with said copy of said hardware token fetched from said HSA for determining if said software definition corresponds with said hardware configuration definition.

L12: Entry 1 of 1 File: USPT Oct 3, 2000

DOCUMENT-IDENTIFIER: US 6128307 A
TITLE: Programmable data flow processor for performing data transfers

Application Filing Date (1):
19971201

Detailed Description Text (136):
FIG. 12 shows a block diagram of the major functional blocks. DFP 320 interfaces to three different bus configurations in the platform; the micro-controller 310, DSP memory 350, and hardware blocks. Each interface contains a functional block to provide buffering to allow interfacing to the stream processor 500 16 bit bus width.

CLAIMS:

1. A programmable data flow processor operable for performing data transfers between a plurality of devices, the data flow processor comprising:

a plurality of ports each for coupling to a device;

programmable configuration registers which are operable to receive data for

programming the data flow processor for data transfers between selected ones of said plurality of ports;

a stream processor coupled to each of said plurality of ports and coupled to said programmable configuration registers, wherein the stream processor is operable to access data comprised in said programmable configuration registers and in response configure a plurality of data streams between

said plurality of ports, wherein each of said plurality of data streams is transferred between at least one source port and at least one destination port, and wherein said stream processor is operable to transfer said plurality of data streams between said plurality of ports;

wherein the data flow processor includes at least one buffer for temporarily storing data transferred between two devices; and

wherein the stream processor determines an ordering of transfers between said plurality of ports based on buffer availability.

5. The data flow processor of claim 2, wherein said programmable configuration registers comprise information indicating the priority of each of said data streams.

15. The data flow processor of claim 1, wherein said programmable configuration registers comprise information regarding which device is interrupted when one of said data streams is complete.

19. A programmable data flow processor operable for performing data transfers, the

data flow processor comprising:

a plurality of ports each for coupling to a device;

programmable configuration registers which are operable to receive data for programming the data flow processor for data transfers between selected ones of said plurality of ports; and

a stream processor coupled to each of said plurality of ports and coupled to said programmable configuration registers, wherein the stream processor is operable to access data comprised in said programmable configuration registers and in response configure a plurality of data streams between said plurality of ports, wherein each of said plurality of data streams is transferred between at least one source port and at least one destination port, and wherein said stream processor is operable to transfer said plurality of data streams between said plurality of ports;

wherein the data flow processor includes at least one buffer for temporarily storing data transferred between two devices;

wherein each of the ports include information indicating availability for data transfer;

wherein the stream processor transfers a programmable amount of data for each of said plurality of data streams, wherein said amount of data is transferred in packets, and wherein an amount of data transferred in each packet is programmable;

wherein the stream processor determines an ordering of transfers between said plurality of ports based on buffer availability and said port availability information; and

wherein the stream processor is operable to determine when data transfers for data streams are to occur.

L3: Entry 7 of 28 File: USPT Sep 15, 1998

DOCUMENT-IDENTIFIER: US 5809176 A
**** See image for Certificate of Correction ****
TITLE: Image data encoder/decoder system which divides uncompresed image data into a plurality of streams and method thereof

Application Filing Date (1):
19951018

Brief Summary Text (34):
More specifically, as each pixel datum forming the image data stream is sent to the arithmetic encoder means of the presently preferred embodiments, it is initially received by the context generator and encoder processor. The context generator generates the reference pixel data for the present pixel datum from portions of the image data stream already received and relays it to the index generator formatted as the context signal CX.

CLAIMS:

8. An image data encoder system, comprising:

a data division circuit to divide image data at least partially representing an image into a plurality of individual image data streams according to at least one of a format of the image data and a spatial relationship between the image data and the image; and

a plurality of dedicated arithmetic encoders in communication with said data division circuit to encode the individual data streams in parallel as a corresponding plurality of coded data streams, each of said dedicated arithmetic encoders corresponding to and encoding one of the individual data streams into one of the coded data streams;

wherein each of said dedicated arithmetic encoders comprises:

context generating means in communication with said data distributor for generating reference pixel data from pixel data contained in the corresponding individual image data stream as a context signal;

index generating means in communication with said context generating means for producing an optimum transition destination index ST.sub.i as a function of the context signal and a transition destination index ST.sub.0 ;

probability estimator means in communication with said index generating means for composing table data corresponding to the optimum transition destination index ST.sub.i, the table data being stored therein and including, for each table index ST, a symbol generating probability for at least one of the group consisting of major and minor symbols, a transition destination index NLPS when the minor symbol is generated, and a transition destination index NMPS when a major symbol is generated;

area register means for storing length data A of a new encoding line created every

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, June 22, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L12	L11 and l6	1
☐	L11	programmable adj4 configuration	2032
☐	L10	programmable adj4 configurator	0
☐	L9	writeable adj4 configurator	0
☐	L8	L7 and l6	1
☐	L7	configuration near5 (spcifier or identifier)	1786
☐	L6	19980508	29
☐	L5	((stream adj4 processor)or multiple adj4 processor or multi-processor) same ((configurable near5 interconnection) or (configuration near5 manager) or (configuration near5 controller))	68
☐	L4	configurable adj4 pin adj4 logic	2
☐	L3	L2 and (stream near5 context)	28
☐	L2	19980508	714
☐	L1	context adj4 processor	1788

END OF SEARCH HISTORY

time pixel data is received;

code register means for storing a coordinate of the point included in the new encoding line created for each pixel data, as position coordinate data C; and

processing means in communication with said index generating means, probability estimator means, said area register means, and said code register means, for each pixel data received by said context generating means:

dividing the encoding line length A and position coordinate data C stored in the area and code registers, respectively, according to symbol generating probability for given pixel data included in the table data of said probability estimator means, in order to create a next new encoding line;

updating the encoding line length A and position coordinate data C stored in the area and code registers, respectively, according to the next new encoding line; and

transmitting to said index generating means one of the input transition destination index NLPS when the given pixel data is a minor symbol and transition destination index NMPS when the given pixel data is a major symbol as the next transition destination index ST0.

10. An image data encoder/decoder system, comprising:

a data division circuit to divide original image data at least partially representing an image into a plurality of individual image data streams according to at least one of a format of the image data and a spatial relationship between the image data and the image;

a plurality of dedicated arithmetic encoders in communication with said first data division circuit to encode the individual data streams in parallel as a corresponding plurality of coded data streams, each of said dedicated arithmetic encoders corresponding to and encoding one of the individual data streams into one of the coded data streams;

a plurality of dedicated arithmetic decoders in communication with said arithmetic encoders, each of said dedicated arithmetic decoders in communication with one of said dedicated arithmetic encoders and decoding a corresponding one of the coded data streams as a decoded data stream; and

a data integrator in communication with said plurality of dedicated arithmetic decoders to combine the decoded data streams and reform the original image data;

wherein each of said dedicated arithmetic decoders comprises:

context generating means in communication with said data encoder for generating reference pixel data for an immediately succeeding code data contained within the corresponding coded data stream based on existing decoded data stream as a context signal;

index generating means in communication with said context generating means for producing an optimum transition destination index ST.sub.i as a function of the context signal and a transition destination index ST.sub.0 ;

probability estimator means in communication with said index generating means for table data corresponding to the optimum transition destination index ST.sub.i, the table data being stored therein and including, for each table index ST, a symbol generating probability for at least one of the group consisting of major and minor symbols, a transition destination index NLPS when the minor symbol is generated,

and a transition destination index NMPS when a major symbol is generated;

area register means for storing length data A of a decoding line;

code register means for storing a coordinate of the point included in the new decoding line as position coordinate data C; and

processing means in communication with said index generating means, probability estimator means, said area register means, and said code register means, for each pixel data received by said context generating means:

for dividing the decoding line specified with the data stored in the area register into major and minor areas according to a symbol generating probability for the pixel data included in the table data of said probability estimator means;

for defining a new decoding line depending on which area a given code data falls in;

for replacing the data stored in said area register with the new length data A;

for outputting the symbol used for selecting said divided area as the decoded pixel data corresponding to the given code data; and

transmitting to said index generating means one of the input transition destination index NLPS when the decoded pixel data is a minor symbol and transition destination index NMPS when the decoded pixel data is a major symbol as the next transition destination index ST0.

First Hit Fwd Refs

Generate Collection

L3: Entry 12 of 28 File: USPT Jul 15, 1997

DOCUMENT-IDENTIFIER: US 5649230 A
**** See image for Certificate of Correction ****
TITLE: System for transferring data using value in hardware FIFO'S unused data start pointer to update virtual FIFO'S start address pointer for fast context switching

Application Filing Date (1):
19950607

Brief Summary Text (22):
However, the use of a deeper FIFO approach means that more instructions and/or data must be dealt with before the stream of instructions and/or data can be switched to another stream of instructions and/or data. Such a switch in the streams of instructions and data is called a "context switch." In other words, all of the instructions and/or data in the FIFO must be dealt with by the I/O controller 106, 112, 116 before the stream of instructions and/or data being provided to the I/O controller 106, 112 or 116 is context switched. This has the effect of increasing latency because of the extra time that is taken before the context switch can take place between streams of instructions and/or data.

Detailed Description Text (7):
As employed herein the term "context" is defined as the current state of a processor. In a CPU the context of that processor would be, for example, the contents of the register files, contents of the status registers, and the current state of the error flags. In the case of a smart peripheral the context attributes might include any attributes that have been set and at what point in its pipeline those attributes have been set. Thus, context switching entails switching the states of the different processes currently running on the computer system employing the present invention. In other words, the states of one process are saved and switched with the states of another process inside the driving virtual FIFO.

Detailed Description Text (54):
Typically, the virtual FIFO (data queue) is swapped when a context switch is performed by the host software operating on the digital processor 126. However, this does not necessarily have to take place. For example, if two processes are associated with the same data queue, such a swap would not take place. An instruction in the data stream would signal the hardware to synchronously switch contexts. A preferred embodiment would not flush the virtual FIFO. Instead, it would just continue reading the data stream. All the instructions ahead of the switch instruction would complete there operations. Subsequently, a set of instructions would be sent down the pipeline to start saving contexts as the different stages become idle. These operations are all synchronous. Thus, it should be understood that the present invention does not require that the data queues be swapped in order for context switching to be performed.

L3: Entry 16 of 28 File: USPT Aug 22, 1995

DOCUMENT-IDENTIFIER: US 5444853 A
TITLE: System and method for transferring data between a plurality of virtual FIFO's and a peripheral via a hardware FIFO and selectively updating control information associated with the virtual FIFO's

Application Filing Date (1):
19920331

Brief Summary Text (22):
However, the use of a deeper FIFO approach means that more instructions and/or data must be dealt with before the stream of instructions and/or data can be switched to another stream of instructions and/or data. Such a switch in the streams of instructions and data is called a "context switch." In other words, all of the instructions and/or data in the FIFO must be dealt with by the I/O controller 106, 112, 116 before the stream of instructions and/or data being provided to the I/O controller 106, 112 or 116 is context switched. This has the effect of increasing latency because of the extra time that is taken before the context switch can take place between streams of instructions and/or data.

Detailed Description Text (7):
As employed herein the term "context" is defined as the current state of a processor. In a CPU the context of that processor would be, for example, the contents of the register files, contents of the status registers, and the current state of the error flags. In the case of a smart peripheral the context attributes might include any attributes that have been set and at what point in its pipeline those attributes have been set. Thus, context switching entails switching the states of the different processes currently running on the computer system employing the present invention. In other words, the states of one process are saved and switched with the states of another process inside the driving virtual FIFO.

Detailed Description Text (54):
Typically, the virtual FIFO (data queue) is swapped when a context switch is performed by the host software operating on the digital processor 126. However, this does not necessarily have to take place. For example, if two processes are associated with the same data queue, such a swap would not take place. An instruction in the data stream would signal the hardware to synchronously switch contexts. A preferred embodiment would not flush the virtual FIFO. Instead, it would just continue reading the data stream. All the instructions ahead of the switch instruction would complete there operations. Subsequently, a set of instructions would be sent down the pipeline to start saving contexts as the different stages become idle. These operations are all synchronous. Thus, it should be understood that the present invention does not require that the data queues be swapped in order for context switching to be performed.

CLAIMS:

9. The system of claim 1, wherein said main memory includes at least one context memory block for storing information concerning the contexts currently operating in the peripheral processor.

First Hit Fwd Refs

Generate Collection

L3: Entry 2 of 28 File: USPT Feb 20, 2001

DOCUMENT-IDENTIFIER: US 6192073 B1
TITLE: Methods and apparatus for processing video data

Application Filing Date (1):
19960819

Brief Summary Text (6):
In some embodiments, the computer system can process several data streams concurrently. As a result, the user of the computer system can have a video conference with two or more parties. Multiple data streams can be processed concurrently because the bitstream processor can switch contexts to encode or decode different data streams concurrently in real time.

Detailed Description Text (11):
Processor 110 can process several data streams at a time. For example, if a user of processor 110 has a video conference with two or more parties, processor 110 provides video and audio processing that allows the user to see and hear the multiple parties. To handle multiple video data streams, processor 110 supports context switching. This means that BP 245 switches between multiple data streams. In a video conference, each data stream may come from a separate remote party. Alternatively, additional data streams may come from movie channels to allow the user to participate in the video conference and watch one or movie presentations at the same time. Context switching is described in Appendix A, Section 10.12. When contexts are to be switched, scalar processor 210 saves the current contexts and initializes BP 245 to process a different context.

L3: Entry 28 of 28 File: USPT Nov 20, 1984

DOCUMENT-IDENTIFIER: US 4484274 A
TITLE: Computer system with improved process switch routine

Application Filing Date (1):
19820907

Brief Summary Text (6):
In order to achieve a high performance system, a processor has to be structured to enable the operating system to switch execution rapidly between individual processes. Since a process is a stream of instructions and data defined by a hardware context, each process has a unique identification in the computer system, the stream of code being executed at any instant being determined by its hardware context. The term "hardware context" refers to the information loaded in the processor's registers that identifies, where the stream of instructions and data are located, which instruction to execute next, and what the processor status is during execution. In an illustrative prior art system (Digital Equipment Corporation VAX-11 computer) the computer's operating system switches between processes by requesting the processor to save one process hardware context and load another via a specific set of instructions.

First Hit Fwd Refs

Generate Collection

L3: Entry 24 of 28 File: USPT Aug 14, 1990

DOCUMENT-IDENTIFIER: US 4949391 A
TITLE: Adaptive image acquisition system

Application Filing Date (1):
19870508

Detailed Description Text (14):
The compressed image data is processed by the character recognition unit 40, along with the system controller 50, to isolate and uniquely identify each image. The token stream for these images is further refined by context processing (processor 48) and translated to a set of output character codes (processor 55) which are sent through the system communication interface 60 to the host computer 100.

Detailed Description Text (38):
At step 164, the function keys 26C-26N are checked and any new function key tokens are output to the queue. At step 166, the image is recognized (character recognition processor 40) and corresponding tokens are output to the queue. At step 168, context processing occurs (processor 48) and at step 170, the resulting character tokens are translated to output character codes (processor 55). Decision point 172 returns operation to step 154 to begin again if all tokens have been queued; otherwise, operation loops back to step 164.

L6: Entry 7 of 29 File: USPT Feb 22, 2000

DOCUMENT-IDENTIFIER: US 6029239 A
TITLE: Configuring a communications system with a configurable data transfer architecture

Application Filing Date (1):
19971201

Detailed Description Text (134):
FIG. 12 shows a block diagram of the major functional blocks. DFP 320 interfaces to three different bus configurations in the platform; the micro-controller 310, DSP memory 350, and hardware blocks. Each interface contains a functional block to provide buffering to allow interfacing to the stream processor 500 16 bit bus width.

L6: Entry 2 of 29 File: USPT Apr 10, 2001

DOCUMENT-IDENTIFIER: US 6216179 B1
TITLE: System for processing an I/O request using an old program while loading a new program which upon completion used to process the I/O request

Application Filing Date (1):
19980128

Brief Summary Text (3):
In the multi-processor system, processing of a job may be efficiently conducted by dividing the job among a plurality of processors for processing. As one form of application of the multi-processor system, a disk controller of a multi-processor configuration has been put into practice. Such a disk controller is disclosed in JP-A-2-62623. In the disk controller disclosed in JP-A-2-62623, a job such as reading or writing of data is distributedly executed by a processor which controls data transfer between a disk controller and a host computer, and a processor which controls data transfer between the disk controller and a disk drive. Specifically, for example, data read from one disk drive by one processor is transferred to the host computer by another processor. Thus, the data transfer process between the host computer and the disk controller and the data transfer process between the disk drive and the disk controller may be executed in parallel and efficient data transfer may be attained.

Brief Summary Text (4):
In the disk controller of the multi-processor configuration described above, it is possible to stop the processing by some processors and continue the data transfer control by the other processors to maintain fault processors. However, in such a maintenance method, it is a premise that the processors which distributedly process a job execute the same version of a program. When versions of the program to be executed by the respective processors are to be upgraded, and the maintenance is conducted in this manner, there occurs a period in which the programs of different versions co-exist and one job may be shared by the processors which execute the different versions of the program. In such a case, there may be no assurance of consistent processing of the job. As a result, when the versions of the program executed by the processors are to be upgraded, the system has to be wholly stopped.

Detailed Description Text (13):
In the data read process by the disk controller of the present embodiment, the process is conducted in the same manner as the data read process in the disk controller of a conventional multi-processor configuration except that each processor refers the execution cluster indication information 600 and the cluster operation mode indication information 700, and refers the executable processor designation information (the executable processor designation bit map 821) and continues the processing while updating. Accordingly, the explanation thereof is omitted.

L6: Entry 4 of 29 File: USPT Oct 3, 2000

DOCUMENT-IDENTIFIER: US 6128307 A
TITLE: Programmable data flow processor for performing data transfers

Application Filing Date (1):
19971201

Detailed Description Text (136):
FIG. 12 shows a block diagram of the major functional blocks. DFP 320 interfaces to three different bus configurations in the platform; the micro-controller 310, DSP memory 350, and hardware blocks. Each interface contains a functional block to provide buffering to allow interfacing to the stream processor 500 16 bit bus width.